PERFECTLY CLEAR API REFERENCE
Android API

Version 7.4.1.0

Background and Overview

Introduction

This document will explain how to use the Perfectly Clear API to enable fully-automatic or interactive image corrections and analysis in your applications.

The API consists to four major components:

1. Perfectly Clear Core Corrections
2. Noise Corrections
3. Red-eye removal
4. “Beautify” Corrections

The first three are provided to all licensees and are always delivered in the libPerfectlyClearPro_v7020_Android.so, while the “Beautify” corrections are optional and are delivered only in libPerfectlyClearPro_fb_v7020_Android.so version. Regardless of presence of the Beautify features the API remains the same. No coding changes are needed to enable these corrections at a later date.

All four correction components include a *Calc* phase, where image analysis is performed, but no corrections are made. Corrections are made in a separate *Apply* phase. The *Calc* phase must be run once per image - and does not change based on image correction parameters, allowing you to speed up the correction results for use in interactive applications where users can alter the correction parameters and getting a preview image back to the customer is the primary goal. In fully automatic correction applications, this isn’t needed, so a single *AutoCorrect* function call will perform both *Calc* and *Apply*.

Reduced Resolution Images in Calc

The Calc function accepts two images as the first two arguments; a full-sized image and a reduced resolution image. The full resolution image is required to run Calc for Noise Removal and Beautify; omitting this parameter will disable Noise and Beautify corrections.

Perfectly Clear Core and Red-eye corrections can run on a reduced resolution image, speeding the Calc processing time without compromising quality. The image must be no smaller than 1024 px on the longer edge, and ideally should be the larger of one-third of the original image or 1024 px (on the longer edge). If a reduced resolution image is not passed into the Calc function, the PFC library will create this for you, using the down-scaling mentioned above.

Color Management

The Perfectly Clear Core corrections assume that the image data is in the sRGB color space. For best image quality, be certain to convert any images in other spaces to sRGB before processing with this API.

Images in Adobe RGB or ProPhoto or other wide-gamut color spaces will appear overly red and overly dark once corrected with Perfectly Clear.

Red Eye Correction

Our red-eye correction technology leads the industry in its speed and accuracy. It automatically detects eyes, determines if "red-eye" is present, and applied a very natural image correction to remove this unsightly camera artifact. As a fully-automatic correction, it can mis-identify red-eyes occasionally. One method to lower the frequency of this is for you to only enable Red-Eye corrections on images where a flash was used – as determined by the EXIF Flash tag. As the libraries provided here only have access to the image content – not the file or EXIF data, you will need to implement this validation yourself.

Usage

You will find functional example code with this API that shows three different manners to use these libraries. The first is the most simple – load and image in memory and auto-correct it in a single function call. The second example shows splitting apart the AutoCorrection separate preCalc and Apply calls, and the third example shows the usage for an interactive application where the correction parameters are altered by the user.

Change Log

Version 7.1

Added support for new SFB parameters:

fb_bSkinToning	BOOL	Set to TRUE to enable skin toning.
fb_iSkinToning	int	Skin Toning level. (0 - 100)
fb_eSkinToningMode	SKINMODE	Use SKINMODE_FACE to apply correction ONLY on skin regions included in faces. Use SKINMODE_BODY to apply correction on most skin regions regardless they are linked with a face or not.
fb_eSkinToningType	SKINTONINGTYPE	See enum definition of SKINTONINGTYPE. P.24
fb_bLipSharpen	BOOL	Set to TRUE to enable lip sharpening.
fb_iLipSharpen	int	Lip sharpening level. (0 - 100)
fb_eLipSharpenType	LIPSHARPENTYPE	See definition of LIPSHARPENTYPE. P.24
fb_bBlush	BOOL	Set to TRUE to add blush.
fb_iBlush	int	Blush level. (0 - 100)

Version 7.2

Added support for Face Aware Exposure.

core_bUseFAE	BOOL	Set to TRUE (recommended) to enable Face Aware Exposure selection. Recommended exposure will be calculated in favor of any human face detected from the image.

New utility functions:

FAEFaceCount

Return number of faces detected during the Face Aware Exposure analysis.

EnumFAEFaceRect

Enumerate faces detected during the FAE analysis.

IsNoiseDetected

Return true if noise detected in noise analysis.

Version 7.3

Added support for format

PFC_PixelFormat32bppARGB
PFC_PixelFormat48bppBGR
PFC_PixelFormat64bppABGR

Added support for opacity in Apply function.

Added preset "Intelligent Auto" support to SetParam().

Added new parameter bRejectMonolith in PFC_Calc().

Added new core error code PFC_CORE_BELOWMINSIZE when image side dimension smaller than 32 pixel.

Added support for variable DCF and Light Diffusion.

fDCF	float	Level of DCF. 0.0 (none applied) to 1.0 (full)
fLightDiffusion	float	Level of light diffusion correction. 0.0 (none applied) to 1.0 (full)

Two parameters changed in name:

bSkinTone is now: bInfrared

fSkinTone is now: fInfrared

Version 7.4

Replace vibrancy processing with a more reliable, gentle correction when value is at zero.

Fix problem with high contrast artifact.

Package:

PerfectlyClear

Interface Class:

V7

Class Function Reference:

CreateEngine

Syntax:

```
ByteBufferCreateEngine();
```

Return:

Engine instance in the form of direct ByteBuffer.

Description:

Create core engine for use in Perfectly Clear image processing pipeline. The Engine instance created by this function can be used in the entire session before DestroyEngine() is called.

DestroyEngine

Syntax:

```
voidDestroyEngine(ByteBuffer Engine);
```

Parameters:

[in]	Engine	Engine instance to be released.

Description:

Release Engine instance created by function CreateEngine() and release all resources used by the Engine instance.

SetParam

Syntax:

```
voidSetParam(PFCPARAM Param, int id);
```

Parameters:

[in/out]	Param	PFCPARAM class instance to receive the parameters.
[in]	id	Preset id PFCPRESETID.

Description:

SetParam initializes a PFCPARAM instance with parameters pertaining to Athentech preset as identified by the PFCPRESETID. If preset id is not provided, the default preset values will be used.

PRESETID	Athentech Preset
PRESET_BEAUTIFY	Beautify
PRESET_BEAUTIFYPLUS	Beautify Plus
PRESET_DETAILS	Details
PRESET_VIVID	Vivid

Calc

Syntax:

```
ByteBufferCalc(int width, int height, int stride, intpixelFormat,
ByteBuffer buffer, intwidthds, intheightds, intstrideds,
ByteBufferbufferds, ByteBuffer Engine);
```

Return:

Profile instance in the form of direct ByteBuffer which contains the analysis results for different features.

Query the Status with function *ProfileStatus()* for high level return code.

0	Success.
> 0	The four least significant bits indicates which feature has not finished successfully.

Bit 0	Problem with Noise Removal.

	Bit 1	Problem with Core analysis.
	Bit 2	Problem with Face Beautification analysis.
	Bit 3	Problem with Red Eye analysis.

Query individual status from the image profile .

PFCNR_Status	Use function NRStatus().
PFCCORE_Status	Use function CORE Status().
PFCFB_Status	Use function FBStatus().
PFCRE_Status	Use function REStatus().

Parameters:

[in]	width	Pixel width of full size image.
[in]	height	Pixel height of full size image.
[in]	stride	Stride value of full size image.
[in]	pixelFormat	Integer value of PFCPIXELFORMAT enum. Use the cardinal() function for translation.
[in]	buffer	Direct ByteBuffer that contains the image data of full size image.
[in]	widthds	Pixel width of down sampled image.
[in]	heightds	Pixel height of down sampled image.
[in]	strideds	Stride value of down sampled image .
[in]	bufferds	Direct ByteBuffer that contains the image data of down sampled image.
[in]	engine	Engine instance returned from CreateEngine().

Description:

Performs initial calculation of image specific profile parameters.

PFC_Apply

Syntax:

```
intApply(int width, int height, int stride, intpixelFormat, ByteBuffer
buffer, ByteBuffer engine, ByteBuffer profile, PFCPARAM param);
```

Return:

<table>
<tr><td>0</td><td>The correction is successful.</td></tr>
<tr><td>> 0</td><td>The return code is composed of four 8 bit sub codes:
<table>
<tr><td>0 - 7</td><td>Returns PFCNR_STATUS Noise Removal correction status.</td></tr>
<tr><td>8 - 15</td><td>Returns PFCCORE_STATUS Core correction status.</td></tr>
<tr><td>16 - 23</td><td>Returns PFCFB_STATUS Face Beautification correction status.</td></tr>
<tr><td>24 - 31</td><td>Returns PFCRE_STATUS Red Eye correction status.</td></tr>
</table></td></tr>
<tr><td>< 0</td><td>Map to PFCAPPLYSTATUS enum.</td></tr>
</table>

Parameters:

[in]	width	Pixel width of image.
[in]	height	Pixel height of image.
[in]	stride	Stride value of image.
[in]	pixelFormat	Integer value of PFCPIXELFORMAT.
[in/out]	buffer	Direct ByteBuffer that carries image data.
[in]	engine	Engine instance from CreateEngine() function.
[in]	profile	Profile instance which is returned from the Calc() function.
[in]	param	PFCParam instance that carries the process parameters.

Description:

Perform correction to image as defined by buffer using user parameters as defined in a PFCPARAM instance. Correction requires profile as calculated by the Calc() function.

ReleaseProfile

Syntax:

```
voidReleaseProfile(ByteBuffer profile);
```

Parameters:

[in]	profile	Direct ByteBuffer of profile instance.

Description:

Release profile instance and its associated resources.

AutoCorrect

Syntax:

```
intAutoCorrect (int width, int height, int stride, intpixelFormat,
ByteBuffer buffer, intwidthds, intheightds, intstrideds,
ByteBufferbufferds, PFCParamparam);
```

Return:

<table>
<tr><td>0</td><td>The correction is successful.</td></tr>
<tr><td>> 0</td><td>Return code is composed of four 8 bit sub codes:
<table>
<tr><td>0 - 7</td><td>Returns PFCNR_STATUS Noise Removal correction status.</td></tr>
<tr><td>8 - 15</td><td>Returns PFCCORE_STATUS Core correction status.</td></tr>
<tr><td>16 - 23</td><td>Returns PFCFB_STATUS Face Beautification correction status.</td></tr>
<tr><td>24 - 31</td><td>Returns PFCRE_STATUS Red Eye correction status.</td></tr>
</table></td></tr>
<tr><td>< 0</td><td>Map to PFCAPPLYSTATUS enum.</td></tr>
</table>

Parameters:

[in]	width	Pixel width of full size image.
[in]	height	Pixel height of full size image.
[in]	stride	Stride value of full size image.
[in]	pixelFormat	Integer value of PFCPIXELFORMAT enum. Use the cardinal() function for translation.
[in/out]	buffer	Direct ByteBuffer that contains the image data of full size image.
[in]	widthds	Pixel width of down sampled image.
[in]	heightds	Pixel height of down sampled image.
[in]	strideds	Stride value of down sampled image.
[in]	bufferds	Direct ByteBuffer that contains the image data of down sampled image.
[in]	param	PFCParam instance

Description:

Composite function that takes an input picture and enhances it base on user parameters. This function encapsulates all the details such as Engine, Profile etc. and is suitable for use in server type mass processing environment. See sample project "Sample 1" for usage details.

AutoCorrectPreset

Syntax:

```
intAutoCorrectPreset(int width, int height, int stride, intpixelFormat,
ByteBuffer buffer, intwidthds, intheightds, intstrideds,
ByteBufferbufferds, intPresetID);
```

Return:

<table>
<tr><td>0</td><td>The correction is successful.</td></tr>
<tr><td>> 0</td><td>Return code is composed of four 8 bit sub codes:
<table>
<tr><td>0 - 7</td><td>Returns PFCNR_STATUS Noise Removal correction status.</td></tr>
<tr><td>8 - 15</td><td>Returns PFCCORE_STATUS Core correction status.</td></tr>
<tr><td>16 - 23</td><td>Returns PFCFB_STATUS Face Beautification correction status.</td></tr>
<tr><td>24 - 31</td><td>Returns PFCRE_STATUS Red Eye correction status.</td></tr>
</table></td></tr>
<tr><td>< 0</td><td>Map to PFCAPPLYSTATUS enum.</td></tr>
</table>

Parameters:

[in]	width	Pixel width of full size image.
[in]	height	Pixel height of full size image.
[in]	stride	Stride value of full size image.
[in]	pixelFormat	Integer value of PFCPIXELFORMAT enum. Use the cardinal() function for translation.
[in/out]	buffer	Direct ByteBuffer that contains the image data of full size image.
[in]	widthds	Pixel width of down sampled image.
[in]	heightds	Pixel height of down sampled image.
[in]	strideds	Stride value of down sampled image.
[in]	bufferds	Direct ByteBuffer that contains the image data of down sampled image.
[in]	PresetID	Integer value of PFCPRESETID enum.

Description:

Composite function that takes an input picture and enhances it using preset parameters specified by PFCPRESETID id.

ApplyLocal

Syntax:

```
intApplyLocal(int width, int height, int stride, intpixelFormat,
ByteBufferbuffer,intxOffset, intyOffset, intwidthOrig, intheightOrig,
ByteBuffer engine, ByteBuffer profile, PFCParamparam);
```

Return:

0	The correction is successful.
> 0	PFCCORE_STATUS enum.
< 0	PFCAPPLYSTATUS enum.

Parameters:

[in]	width	Pixel width of full size image.
[in]	height	Pixel height of full size image.
[in]	stride	Stride value of full size image.
[in]	pixelFormat	Integer value of PFCPIXELFORMAT enum. Use the cardinal() function for translation.
[in/out]	buffer	Direct ByteBuffer that contains the image data of full size image.
[in]	xOffset	Horizontal (x) offset of image segment defined in pImage with respect to the original image.
[in]	yOffset	Vertical (y) offset of image segment defined in pImage with respect to the original image.
[in]	widthOrig	Pixel width of the original image.
[in]	heightOrig	Pixel height of the original image.
[in]	engine	Engine returned from CreateEngine() function.
[in]	profile	Profile which is returned from the Calc() function.
[in]	param	PFCParam instance that carries the process parameters.

Description:

Perform CORE correction only to partial image using user parameters as defined in a PFCParam structure. Correction requires profile as calculated by the Calc() function.**IMPORTANT: Only the core correction is applied.**

HasFaceBeautification

Syntax:

```
booleanHasFaceBeautification(ByteBuffer engine);
```

Return:

True if feature is available for use (using libPerfectlyClearPro_fb_v7010.so). False otherwise (using libPerfectlyClearPro_v7010.so).

Parameters:

[in]	engine	ByteBuffer instance returned from CreateEngine() function.

Description:

Utility function to query if Face Beautification feature is available at run time. This usually

FBFaceCount

Syntax:

```
intFBFaceCount(ByteBuffer profile);
```

Return:

Number of face(s) detected.

Parameters:

[in]	profile	Profile instance which is returned from the Calc() function.

Description:

Utility function to query the number of faces detected during Face Beautification analysis.

GetFaceInfo

Syntax:

```
booleanGetFaceInfo(ByteBuffer profile, FaceInfo face, int index);
```

Return:

True if face information is retrieved successfully. False if face is not detected or index is out of bound.

Parameters:

[in]	profile	Profile returned from Calc() function.
[out]	face	FaceInfo instance that carries face and eye details upon successful retrieval. See FaceInfo.java for details.
[in]	index	Index navigating the list. Must be >=0 and < number of faces detected.

Description:

Utility function to query geometry of detected faces.

Example:

```
// Find out number of faces
intfaceCount = m_Pfc.FBFaceCount(m_Profile);
if (faceCount> 0) {
      for(inti = 0; i<faceCount; i++)
      {
            FaceInfo info = new FaceInfo();
            m_Pfc.GetFaceInfo(m_Profile, info, i);
            m_StatusView.setText(Integer.toString(faceCount) + ": " +
Integer.toString(info.face_left)+","+Integer.toString(info.face_top));
      }
}
```

AbnormalTintDetected

Syntax:

```
booleanAbnormalTintDetected(ByteBuffer profile, intTintMethod);
```

Return:

True if abnormal tint is detected when tint detection mode is TintMethod. False otherwise.

Parameters:

[in]	profile	Profile returned from Calc() function.
[in]	TintMethod	Integer value of TINTCORRECTION enum to specify tint detection method used.

Description:

Utility function to query if abnormal tint is detected at certain detection mode.

ProfileStatus

Syntax:

```
intFBFaceCount(ByteBuffer profile);
```

Return:

Return code from Calc.

0	Success.
> 0	The four least significant bits indicates which feature has not finished successfully.

Bit 0	Problem with Noise Removal.
Bit 1	Problem with Core analysis.
Bit 2	Problem with Face Beautification analysis.
Bit 3	Problem with Red Eye analysis.

Parameters:

[in]	profile	Profile instance which is returned from the Calc() function.

Description:

Utility function to query the number of faces detected during Face Beautification analysis.

NRStatus

Syntax:

```
intNRStatus(ByteBuffer profile);
```

Return:

PFCNRSTATUS.

Parameters:

[in]	profile	Profile instance which is returned from the Calc() function.

Description:

Retrieve return code specific to the noise analysis within Calc process.

CORE Status

Syntax:

```
intCORESTatus(ByteBuffer profile);
```

Return:

PFCCORE_STATUS.

Parameters:

[in]	profile	Profile instance which is returned from the Calc() function.

Description:

Retrieve return code specific to the core Perfectly Clear analysis within Calc process.

FBStatus

Syntax:

```
intFBStatus(ByteBuffer profile);
```

Return:

PFCFB_STATUS.

Parameters:

[in]	profile	Profile instance which is returned from the Calc() function.

Description:

Retrieve return code specific to the Face Beautification analysis within Calc process.

REStatus

Syntax:

```
intREStatus(ByteBuffer profile);
```

Return:

PFCRE_STATUS.

Parameters:

[in]	profile	Profile instance which is returned from the Calc() function.

Description:

Retrieve return code specific to the Red Eye detection within the Calc process.

Parameter Class

PFCParam

Parameter group for Perfectly Clear Core correction.

Parameter	Type	Description	Range
core_bEnabled	BOOL	Set to true to enable the entire Core correction.	
core_bAbnormalTintRemoval	BOOL	Set to true to enable Abnormal Tint Removal. Recommended default is FALSE.	
core_eTintMode	enum TINTCORRECTION	ENUM value defined in TINTCORRECTION. It sets the aggressiveness of Tint Removal.	
core_fTintScale	float	Scalar value of how much tint correction should be applied.	0.0 to 1.0
core_iBlackEnhancement	Int	Set luminance threshold for Noise Management	0 to 25 (12)
core_bVibrancy	BOOL	Set to true (recommended default) to enable Color Vibrancy in the library.	
core_iVibrancy	int	Degree of color vibrancy. This value will only be use when bVibrancy is TRUE. Very large values can produce extreme adjustments, so a value of 0, 5, or perhaps a high as 10 is advised.	0 to 200 (5)
core_iStrength	int	Set the strength of exposure correction. If Automatic Strength Selection is enabled, the recommended value is put in this variable upon function return.	0 to 150 (100)
core_bContrast	BOOL	Set to TRUE to also apply Athentech‟s patented Medical Imaging contrast technology.	
core_eContrastMode	Enum CONTRASTMODE	Select contrast mode.	
core_iContrast	int	Intensity of contrast or depth	0 to 100
core_eBiasMode	Enum BIASMODE	Skin and depth bias control. Recommended value is BIAS_AVERAGE_PREFERENCE, unless you are printing to an indigo, iGen, or NexPress printer. If this is the case then use	

		BIAS_BRIGHTER_PREFERENCE.	
core_fBiasScale	float	Scalar value of how much BIAS correction should be applied.	0.0 to 1.0
core_bSharpen	BOOL	Set to TRUE to enable sharpening.	
core_fSharpenScale	float	Sharpening intensity. This value controls how much sharpening to be applied.	0.0 to 3.0 (0.6)
core_bUseAutomaticStrengthSelection	BOOL	Set to TRUE (recommended default) to enable Automatic Strength Selection. Perfectly Clear will determine the optimum strength required for the input image. The value originally passed to the library in iStrength will be ignored. The strength recommended by Automatic Strength Selection will be put in iStrength upon return to caller.	
core_bUseFAE	BOOL	Set to TRUE (recommended) to enable Face Aware Exposure selection. Recommended exposure will be calculated in favor of any human face detected from the image.	
core_eAggressiveness	enum AGGRESSIVENESS	The desired level of lighting for Automatic Strength Selection to target at the Exposure Correction.	
core_iMaxStrength	int	Use this value to limit the maximum Exposure to be applied on the Automatic Exposure Strength Selection algorithm.	0 to 150 (100)
core_bSkinTone	BOOL	Set to TRUE (recommended default) to enable infrared correction. It‟s a good idea to turn Infrared Correction on if correcting lots of people in the picture.	
core_fSkinTone	float	Scale value to control intensity of infrared correction.	0.0 to 1.0
core_bLightDiffusion	BOOL	Set to TRUE to enable light diffusion during DCF correction.	
core_fLightDiffusion	float	Scale value to control intensity of light diffusion correction.	0.0 to 1.0
core_bDCF	BOOL	Set to TRUE to enable Digital Color Fidelity. Recommended value is FALSE.	
core_eDCFMode	enum DCFMODE	Select different class of DCF.	

core_fDCF	float	Scale value to control intensity of Digital Color Fidelity correction.	0.0 to 1.0
bDynamicRange	BOOL	Set to TRUE to enable dynamic range correction. In Athentech's own software, we set this to TRUE when bContrast is true or when any exposure correction is used (bUseAutomaticStrengthSelection = true or iStrenght> 0). This ensures that no "exposure" or dynamic range shift happens if you are only using this PfC correction for 'Beautify' corrections or other sets of corrections that should not change the exposure of the photo.	

Parameter group for Face Beautification.

fb_bEnabled	BOOL	Set to TRUE to enable entire face beautification.	
fb_bFaceSmooth	BOOL	Set to TRUE to enable face smoothing.	
fb_iFaceSmoothLevel	int	Face smoothing level. (0 - 100)	
fb_iSmoothMode	int	Use 1 to apply correction ONLY on skin regions included in faces. Use 2 to apply correction on most skin regions regardless they are linked with a face or not.	
fb_iSmoothType	int	1	The subtle correction removes the wrinkles and spots alone while it keeps the texture of the face unchanged.
		2	This type of correction provides a more uniform appearance to the complexion. It combines the natural look of the SUBTLE mode

<table>
<tr><td></td><td></td><td><table><tr><td></td><td>with the improved efficiency of SUPERSMOOTH.</td></tr><tr><td>3</td><td>This is a more aggressive and effective correction where the appearance of the entire skin (including wrinkles, spots, hot spots) is changed (softened).</td></tr></table></td></tr>
<tr><td>fb_iEyeEnlargeLevel</td><td>int</td><td>Eye enlargement level.</td></tr>
<tr><td>fb_bEyeEnhance</td><td>BOOL</td><td>Set to TRUE to enable eye enhancement.</td></tr>
<tr><td>fb_iEyeEnhanceLevel</td><td>int</td><td>Eye enhancement level.</td></tr>
<tr><td>fb_bEyeCircleRemoval</td><td>BOOL</td><td>Set to TRUE to enable eye circle removal.</td></tr>
<tr><td>fb_iEyeCircleRemovalLevel</td><td>int</td><td>Eye circle removal level. (0 - 100)</td></tr>
<tr><td>fb_bTeethWhiten</td><td>BOOL</td><td>Set to TRUE to enable teeth whitening.</td></tr>
<tr><td>fb_iTeethWhitenLevel</td><td>int</td><td>Teeth whitening level. (0 - 100)</td></tr>
<tr><td>fb_bBlemishRemoval</td><td>BOOL</td><td>Set to TRUE to enable blemish removal.</td></tr>
<tr><td>fb_iBlemishRemovalLevel</td><td>int</td><td>Blemish removal level. (0 - 100)</td></tr>
<tr><td>fb_bFaceSlim</td><td>BOOL</td><td>Set to TRUE to enable face slimming.</td></tr>
<tr><td>fb_iFaceSlimLevel</td><td>int</td><td>Face slimming level. (0 - 100)</td></tr>
<tr><td>fb_bDeFlash</td><td>BOOL</td><td>Set to TRUE to enable deflash.</td></tr>
<tr><td>fb_iDeFlashLevel</td><td>int</td><td>Deflash level. (0 - 100)</td></tr>
<tr><td>fb_bCatchLight</td><td>BOOL</td><td>Set to TRUE to enable catchlight removal.</td></tr>
<tr><td>fb_iCatchLight</td><td>int</td><td>Catchlight level. (0-100)</td></tr>
<tr><td>fb_bSkinToning</td><td>BOOL</td><td>Set to TRUE to enable skin toning.</td></tr>
<tr><td>fb_iSkinToning</td><td>int</td><td>Skin Toning level. (0 - 100)</td></tr>
<tr><td>fb_iSkinToningMode</td><td>int</td><td>Use 1 to apply correction ONLY on skin regions included in faces. Use 2 to apply correction on most skin regions regardless they are linked with a face or not.</td></tr>
<tr><td>fb_iSkinToningType</td><td>int</td><td><table><tr><td>1</td><td>Whitens (bleaches) face. Recommended mainly for darker skin.</td></tr><tr><td>2</td><td>Makes skin look lighter and more pale.</td></tr><tr><td>3</td><td>Warms skin tone.</td></tr><tr><td>4</td><td>Darkens skin, makes it look naturally tanned.</td></tr><tr><td>5</td><td>Adjust skin to user defined foundation color.</td></tr></table></td></tr>
<tr><td>fb_bLipSharpen</td><td>BOOL</td><td>Set to TRUE to enable lip sharpening.</td></tr>
</table>

<table>
<tr><td>fb_iLipSharpen</td><td>int</td><td colspan="2">Lip sharpening level. (0 - 100)</td></tr>
<tr><td>fb_eLipSharpenType</td><td>int</td><td colspan="2"><table><tr><td>1</td><td>Fine touch of sharpening.</td></tr><tr><td>2</td><td>Stronger sharpening. Details are more pronounced.</td></tr><tr><td>3</td><td>Lip details are coarsely pronounced.</td></tr></table></td></tr>
<tr><td>fb_bBlush</td><td>BOOL</td><td colspan="2">Set to TRUE to add blush.</td></tr>
<tr><td>fb_iBlush</td><td>int</td><td colspan="2">Blush level. (0 - 100)</td></tr>
</table>

Parameter group for Noise Removal.

nr_bEnabled	BOOL	Set to TRUE to enable noise removal.
nr_iPreset	int	Set preset number. 0 - default 1 - portrait 2 - night 3 - cameraphone 4 - force noise removal
nr_iStrengthOffset	int	Set offset to recommended level of noise removal strength. Range from -5 to +5. Set to 0 to use recommended value.
nr_iDetailOffset	int	Set offset to recommended level of preservation of details. Range from -30 to +30. Set to 0 to use recommended value.

Parameter group for Red Eye correction.

re_bEnabled	BOOL	Set to TRUE to enable red eye removal.

FaceInfo

face_left	int	Horizontal coordinate of left side of the rectangle.
face_top	Int	Vertical coordinate of top of the rectangle.
face_width	Int	Width of rectangle.
face_height	Int	Height of rectangle.

leftEye_x	int	X coordinate of left eye.
leftEye_y	int	Y coordinate of left eye.
rightEye_x	int	X coordinate of right eye.
rightEye_y	int	Y coordinate of right eye.

Enums

PFCPIXELFORMAT

PFC_PixelFormat24bppRGB	24 bits per pixel; 8 bits each are used for the red, green, and blue components. Byte order [BGRBGR]
PFC_PixelFormat24bppBGR	24 bits per pixel; 8 bits each are used for the red, green, and blue components. Byte order [RGBRGB]
PFC_PixelFormat32bppABGR	32 bits per pixel; 8 bits each are used for the alpha, red, green, and blue components. Byte order [RGBARGBA]
PFC_PixelFormat48bppRGB	48 bits per pixel; 16 bits each are used for the red, green, and blue components. Byte order [BBGGRRBBGGRR]
PFC_PixelFormat64bppARGB	64 bits per pixel; 16 bits each are used for the alpha, red, green, and blue components. Byte order [BBGGRRAABBGGRRAA]
PFC_PixelFormat32bppARGB	32 bits per pixel; 8 bits each are used for the alpha, red, green, and blue components. Byte order [BGRABGRA]
PFC_PixelFormat48bppBGR	48 bits per pixel; 16 bits each are used for the red, green, and blue components. Byte order [RRGGBBRRGGBB]
PFC_PixelFormat64bppABGR	64 bits per pixel; 16 bits each are used for the alpha, red, green, and blue components. Byte order [RRGGBBAARRGGBBAA]

PFCPRESETID

PRESET_BEAUTIFY	Beautify
PRESET_BEAUTIFYPLUS	Beautify Plus
PRESET_DETAILS	Details
PRESET_VIVID	Vivid
PRESET_INTELLIGENTAUTO	Intelligent Auto

TINTCORRECTION

TINTCORRECT_AGGRESSIVE	Aggressive detection of tint.
TINTCORRECT_DEFAULT	Moderate level of tint detection.
TINTCORRECT_CONSERVATIVE	Priority on minimum false positive detection.
TINTCORRECT_STRONGEST	Highest sensitivity level of tint detection.

CONTRASTMODE

HIGH_CONTRAST	Optimized to bring higher contrast to the image.
HIGH_DEFINITION	Optimized to bring out more details in the shadows, more details in the highlights, and more pleasing skin tones.

DCFMODE

DCF_STANDARD	For normal photo.
DCF_VIVID	For more color vibrancy.

AGGRESSIVENESS

AGGRESSIVENESS_CONSERVATIVE	Less aggressive in exposure correction.
AGGRESSIVENESS_MODERATE	Moderate level of exposure correction.
AGGRESSIVENESS_AGGRESSIVE	More aggressive in exposure correction.

BIASMODE

BIAS_NONE	Turn off bias correction.
BIAS_ASIAN_PREFERENCE	Fine tuned for Asian skin tone.
BIAS_AVERAGE_PREFERENCE	For average usage.
BIAS_BRIGHTER_PREFERENCE	Average usage with brighter tone.

PFCNR_STATUS

PFC_NR_SUCCESS	Success.
PFC_NR_NOTENABLED	Feature not enabled.
PFC_NR_FULLRES_REQUIRED	Full res image (pImage) is missing.
PFC_NR_CANCELLED	Process cancelled.
PFC_NR_ERRBITMAP	Error reading image data.
PFC_NR_ERRSETTINGS	Error in settings.
PFC_NR_MISC_ERROR	Misc. errors.
PFC_NR_NOTFOUND	Noise not found.

PFCCORE_STATUS

PFC_CORE_SUCCESS	Success.
PFC_CORE_NOTENABLED	Feature not enabled.
PFC_CORE_CANCELLED	Process cancelled.
PFC_CORE_NOSOURCEIMAGE	Full res source image (pImage) is missing.
PFC_CORE_INSUFFICIENTMEMORY	Process aborted because of insufficient memory.

PFCFB_STATUS

PFC_FB_SUCCESS	Success.
PFC_FB_NOTENABLED	Feature not enabled.

PFC_FB_WARNING	Warning. e.g. face not detected.
PFC_FB_FULLRES_REQUIRED	Full res image (pImage) is missing.
PFC_FB_CANCELLED	Process cancelled.
PFC_FB_FUNCTION_ERROR	Unable to locate function in the SFBEngine library.
PFC_FB_CREATE_ENGINE_FAILED	Unable to create SFB Engine object for processing.
PFC_FB_ANALYSIS_FAILED	The face analysis did not complete successfully.
PFC_FB_NO_CORRECTION	No correction occur during process.
PFC_FB_NOT_EXECUTED	Not executed.
PFC_FB_NOT_AVAILABLE	Face beautification feature not available.

PFCRE_STATUS

PFC_RE_SUCCESS	Success.
PFC_RE_NOTENABLED	Feature not enabled.
PFC_RE_FULLRES_REQUIRED	Full res image (pImage) is missing.
PFC_RE_NOT_FOUND	Red eye not found.
PFC_RE_GEN_ERROR	General error.
PFC_RE_INVALID_PARAMETER	Invalid parameter.
PFC_RE_NO_MEMORY	Insufficient memory.
PFC_RE_CANCELLED	Process cancelled.
PFC_RE_NOT_SUPPORTED	Not supported.

Usage & Examples

Scenerio #1 - Using AutoCorrect

The simplest way to use Perfectly Clear library suite. This protocol is more suitable for developing a server type software project.

1. Instantiate V7 class insatnce:

```
V7 Pfc = new V7();
```

2. Initialize the parameter structure:

```
PFCParamparam = new PFCParam();

Pfc.SetParam(param, 0);
```

3. Perform full correction using the auto function:

```
int ret = Pfc.AutoCorrect(w, h, w * 4,
V7.PFCPIXELFORMAT.PFC_PixelFormat32bppABGR.ordinal(), bb, w1, h1, w1 *
4, bbds, param);
```

Example:

```
public void onClickTest(View view) {

Bitmap bm = BitmapFactory.decodeResource(getResources(), R.drawable.redeye);

Bitmap bmm = bm.copy(Bitmap.Config.ARGB_8888, true);

BitmapFactory.Options options = new BitmapFactory.Options();

options.inSampleSize = 2;

Bitmap bmds = BitmapFactory.decodeResource(getResources(), R.drawable.redeye, options);

Bitmap bmmds = bmds.copy(Bitmap.Config.ARGB_8888, true);

int w, h;

w = bmm.getWidth();

h = bmm.getHeight();

m_Debug.setText("w,h " + Integer.toString(w) + "," + Integer.toString(h));

int[] buffer = new int[w * h];

bmm.getPixels(buffer, 0, w, 0, 0, w, h);

ARGB2RGBA(buffer);

ByteBuffer bb = ByteBuffer.allocateDirect(w * h * 4);
```

```
IntBufferintbuf = bb.asIntBuffer();
intbuf.put(buffer);

int w1, h1;
w1 = bmmds.getWidth();
h1 = bmmds.getHeight();
int[] bufds = new int[w1 * h1];
bmmds.getPixels(bufds, 0, w1, 0, 0, w1, h1);
ARGB2RGBA(bufds);
ByteBufferbbds = ByteBuffer.allocateDirect(w1 * h1 * 4);
IntBufferintbufds = bbds.asIntBuffer();
intbufds.put(bufds);
// Instantiate V7 class instance
V7 Pfc = new V7();
// Initialize PFCParam
PFCParamparam = new PFCParam();
Pfc.SetParam(param, 0);
// Use AutoCorrect() for image enhancement
int ret = Pfc.AutoCorrect(w, h, w * 4,
V7.PFCPIXELFORMAT.PFC_PixelFormat32bppABGR.ordinal(), bb, w1, h1, w1 * 4, bbds, param);
/*
**  Alternatively, you can use AutoCorrectPreset and simply specify the preset ID
**
**     int ret = m_Pfc.AutoCorrectPreset(w, h, w * 4, 3, bb, w1, h1, w1 * 4, bbds, 0);
*/
m_Debug.setText("Return code = " + Integer.toString(ret));
intbuf.rewind();
try {
intbuf.get(buffer);
}
catch(Exception ex) {
String err = ex.getMessage();
}
```

```
RGBA2ARGB(buffer);
bmm.setPixels(buffer, 0, w, 0, 0, w, h);
m_Display.setImageBitmap(bmm);
m_Display.invalidate();
}
```

Scenerio #2 - Separate Calc and Apply

More advanced way to use Perfectly Clear library suite.

1. Instantiate V7 class instance.

```
PFCPARAM param;
```

2. Create process engine:

```
ByteBuffer engine = Pfc.CreateEngine();
```

3. Initialize the parameter class (using default parameter values):

```
PFCParamparam = new PFCParam();

Pfc.SetParam(param, 0);
```

4. Perform pre-calculation of image specific profile:

```
ByteBuffer Profile = Pfc.Calc(w, h, w * 4,
V7.PFCPIXELFORMAT.PFC_PixelFormat32bppABGR.ordinal(), bb, w1, h1, w1 *
4, bbds, Engine);
```

(pEngine is created from step 2.)

5. Apply the calculated profile and parameters to the image.

```
int ret = Pfc.Apply(w, h, w * 4,
V7.PFCPIXELFORMAT.PFC_PixelFormat32bppABGR.ordinal(), bb, Engine,
Profile, param);
```

6. Release resources used by PROFILE:

```
Pfc.ReleaseProfile(profile);
```

7. Release the engine to release resource:

```
Pfc.DestroyEngine(engine);
```

Example:

```
public void onClickTest(View view) {

Bitmap bm = BitmapFactory.decodeResource(getResources(), R.drawable.testimage);

Bitmap bmm = bm.copy(Bitmap.Config.ARGB_8888, true);

BitmapFactory.Options options = new BitmapFactory.Options();
```

```
options.inSampleSize = 2;

Bitmap bmds = BitmapFactory.decodeResource(getResources(), R.drawable.testimage,
options);

Bitmap bmmds = bmds.copy(Bitmap.Config.ARGB_8888, true);

int w, h;

w = bmm.getWidth();

h = bmm.getHeight();

m_Debug.setText("w,h " + Integer.toString(w) + "," + Integer.toString(h));

int[] buffer = new int[w * h];

bmm.getPixels(buffer, 0, w, 0, 0, w, h);

ARGB2RGBA(buffer);

ByteBuffer bb = ByteBuffer.allocateDirect(w * h * 4);

IntBufferintbuf = bb.asIntBuffer();

intbuf.put(buffer);


int w1, h1;

w1 = bmmds.getWidth();

h1 = bmmds.getHeight();

int[] bufds = new int[w1 * h1];

bmmds.getPixels(bufds, 0, w1, 0, 0, w1, h1);

ARGB2RGBA(bufds);

ByteBufferbbds = ByteBuffer.allocateDirect(w1 * h1 * 4);

IntBufferintbufds = bbds.asIntBuffer();

intbufds.put(bufds);


V7 Pfc = new V7();

// Step 1. Create PFC engine

ByteBuffer Engine = Pfc.CreateEngine();

// Step 2. Analyze image and generate image specific profile

ByteBuffer Profile = Pfc.Calc(w, h, w * 4,
V7.PFCPIXELFORMAT.PFC_PixelFormat32bppABGR.ordinal(), bb, w1, h1, w1 * 4, bbds,
Engine);
```

```
// Step 3. Create parameter instance and set values to use those of PRESET_DETAILS (id
0)

PFCParamparam = new PFCParam();

Pfc.SetParam(param, 0);

// Step 4. Enhance image with user parameters

int ret = Pfc.Apply(w, h, w * 4, V7.PFCPIXELFORMAT.PFC_PixelFormat32bppABGR.ordinal(),
bb, Engine, Profile, param);

m_Debug.setText("Return code = " + Integer.toString(ret));

Pfc.ReleaseProfile(Profile);

Pfc.DestroyEngine(Engine);

intbuf.rewind();

try {

intbuf.get(buffer);

}

catch(Exception ex) {

String err = ex.getMessage();

}

RGBA2ARGB(buffer);

bmm.setPixels(buffer, 0, w, 0, 0, w, h);

m_Display.setImageBitmap(bmm);

m_Display.invalidate();

}
```

Sample Projects (Android version)

Three sample Android Studio projects are provided to showcase the usage of the API.

Sample 1

The Basic Sample demonstrates the most basic usage of the API. The sample application performs very basic image processing work flow: input, process, output.

Sample 2

Similar to the basic sample, the sample shows the simple way of using the API but unlike the basic sample, sample 1 gives user more controls on return code handling. The sample first analyses the input image with Calc(). The results (PROFILE) is used in Apply() to carry out actual image enhancement.